Pilha (= stack) e sua API

- Uma *pilha* (= *stack*) é um ADT que consiste em uma coleção de coisas munida de duas operações: *push*, que insere uma coisa na coleção, e *pop*, que remove a coisa mais recente da coleção. (Portanto, a última coisa a entrar é a primeira a sair – LIFO)
- As coisas de que uma pilha é feita serão chamadas *itens*. Os itens podem ser números, strings, structs, etc., etc.
- Exemplos de clientes de pilhas: botão voltar de um browser, cálculo do valor de uma expressão aritmética.

Pilha: API (= Interface)

- Pilha de itens (além das operações fundamentais push e pop, temos algumas operações auxiliares):

public class Stack<Item>

Stack()	construtor: cria uma pilha de Itens vazia
void push(Item item)	insere item nesta pilha
Item pop()	remove o Item mais recente desta pilha
boolean isEmpty()	esta pilha está vazia?
int size()	número de Itens nesta pilha

Item é um *tipo genérico*, ou *parâmetro de tipo*, que deve ser substituído por um tipo concreto quando uma instância da pilha é criada.

Pilha: Implementação

- Exemplo de cliente de pilha: lê strings da entrada padrão e insere-as numa pilha, a menos que a string lida seja "-", caso em que remove uma string da pilha
- Pilha implementada em vetor com redimensionamento
- Pilha implementada em lista ligada (Extra)

Pilha: Exercício

- Escreva um programa que leia linhas de texto da entrada padrão (ignore as linhas vazias) e coloque as linhas em ordem alfabética. As linhas de texto têm comprimento arbitrário e o número de linhas é arbitrário.
- Sugestões:
  - Use a classe *JOpitionPane* para entrada de dados.
  - Construa um vetor de strings usando redimensionamento.
  - Use o método java.util.Arrays.sort() para colocar o vetor em ordem lexicográfica. Depois, repita os testes usando Insertion.sort(), Merge.sort(), e Quick.sort().

Fila(= queue) e sua API

- Uma *fila*(= *queue*) é um ADT que consiste em uma coleção de coisas munida de duas operações: *enqueue*, que insere uma coisa na coleção, e *dequeue*, que remove a coisa mais antiga da coleção. (Portanto, a primeira coisa a entrar é a primeira a sair – FIFO)
- As coisas de que uma fila é feita serão chamadas *itens*. Os itens podem ser números, strings, structs, etc., etc.
- Exemplo: fila de pessoas esperando por atendimento no caixa de um banco.

Fila: API (= Interface)

- Fila de itens (além das operações fundamentais enqueue e dequeue, temos algumas operações auxiliares):

  public class Queue<Item>

Queue()	construtor: cria uma fila de Itens vazia
void enqueue(Item item)	coloca item nesta fila
Item dequeue()	remove o Item mais antigo desta fila
boolean isEmpty()	esta fila está vazia?
int size()	número de Itens nesta fila

Fila: Implementação

- Exemplo de cliente de fila: cálculo da distância entre dois vértices de em grafo (busca em largura).

- Fila implementada em lista ligada (Extra)

Fila: Exercícios

- Exercício 1: escrever um exemplo de cliente para fila análogo ao que usamos para pilha.

- Exercício 2: Imprimir as últimas *k* linhas de um arquivo (grande) de texto. Faça isso sem desperdiçar memória. Use uma (ou mais) das ADTs que já estudamos.

Saco(= bag) e sua API

- Um *saco*(= *bag*) é um ADT que consiste em uma coleção de coisas munida de duas operações: *add*, que insere uma coisa na coleção, e *iterate*, que percorre as coisas da coleção (ou seja, examina as coisas uma a uma).
- A ordem em que o iterador percorre as coisas não é especificada e está fora do controle do cliente.
- Como de hábito, as coisas de que um saco é feito serão chamadas *itens*.
- Diferentemente de um conjunto matemático, os itens de um saco não precisam ser distintos dois a dois.

Saco: API (= Interface)

public class Bag<Item> implements Iterable<Item>

Bag()	construtor: cria um saco de Itens vazio
void add(Item item)	coloca item neste saco
Iterator<Item> iterator()	um iterador que percorre os Itens do saco
boolean isEmpty()	este saco está vazio?
int size()	número de Itens neste saco

Saco: API (= Interface)

- O método iterator() implementa a operação *iterate*: para processar, um a um, todos os itens de uma instância de saco (Bag), um cliente deve dizer:

```
Iterator<Item> it = saco.iterator();
while (it.hasNext()) {
    Item x = it.next();
     ... // processamento de x
}
```

- Felizmente, o cliente pode abreviar esse bloco de código por uma instrução muito simples conhecida como *foreach*:

```
for (Item x : saco) {
    ... // processamento de x
}
```

OBS: Leia para cada x no saco faça ....

Saco: Implementação

- Exemplo de cliente de saco: calcular a *média* e o *desvio padrão* de números dados na entrada padrão.
- Saco implementado em lista ligada (Extra)
- O saco bag armazena os dados, permitindo que eles sejam examinados mais de uma vez. Sem isso, o problema seria mais difícil de resolver.